The Pennsylvania State University

The Graduate School

Capital College

A System to Generate a Simple and Reusable Web-enabled Solution For Database Queries

A Master's paper in

Computer Science

By

Qingru Zhang

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

February 2000

Abstract

Database administrators have to issue many repetitive and tedious database queries every day. They want software that can reduce this redundant work. Furthermore, this software should be portable to as many database systems as possible and should be usable by non-professional users. This paper describes a system that can generate such software automatically. The users of this system define their queries in a text file using some simple formats, and then this system reads this text file and generates the corresponding web-enabled program to implement these queries on the Internet. In this sense, this system is a compiler. The generating system and target program are developed in Perl.

The target web-enabled program can easily be used with a variety of database systems. Due to the interoperability and reusability of Internet components, it is also reusable and easy to use.

Table of contents

Abstract 2
Table of contents 3
Table of Figures 3
1. Introduction 4
2. System Description and Implementation 7
<2.1> System Description from the user view. 7
<2.2> System implementation from the programmer's view. 18
3. Conclusion 29
References 32

Table of Figures

Figure 1. Example text file. 9
Figure 2 Create table statements 12
Figure 3. Login page. 14
Figure 4. Table and query selection page 14
Figure 5. Table value input page. 15
Figure 6. Query result without dynamic variable value input. 16
Figure 7. Variable value entry page for query. 16
Figure 8. Query result with dynamic variable value needed 17
Figure 9. Error message page. 17
Figure 10. Diagram for our overall system. 28

1. Introduction

Electronic commerce is rapidly growing in the current information technology (IT) field. It has been estimated that Internet based business will be the main trade method in the world in the near future [Coen98]. The reason is that Internet components have several advantages, including integration, interoperability and reusability, and the most important: worldwide accessibility. E-commerce is such a fast growing market that many computer companies are engaging in providing all kinds of the web-enabled solutions for industries. The typical products are IBM's customer relationship management (CRM) solutions that are designed to meet various business needs and e-business objectives across different industries.

This paper introduces a specific generating system that generates a web-enabled solution for database queries. This generating system first reads a text file that defines the table and queries in a simple format, then generates a target CGI (Common Gateway Interface) program. The target program provides a user-friendly interface to run the queries on the Internet. Hence, the target program simplifies the database query work for database administrators and makes these queries reusable on the Internet. The main advantages of our system are reusability, portability and simplicity.

The original idea for our overall system came from database administrators (DBAs). They complain that they have to issue hundreds of database queries every day. Additionally, they create many tables that have similar structures, and insert many records into these tables each day. They also do many queries corresponding to these tables and values. This work is repetitive and tedious. Thus, they want software that can reduce this redundant work.

Furthermore, this software needs to be portable to as many database systems as possible. Another issue considered is simplicity. Not only database administrators, but also most of the non-professional users should be able to operate the system. Thus, it frees the professional from redundant work.

Our overall system is designed to meet all of these requirements. First, the generating system generates a web CGI script to run queries on the Internet. Because of the interoperability and reusability of the Internet components, this CGI solution program enables the repeated works to be defined once and then reused on the Internet. It also provides a friendly interface for inserting the values into the tables. Secondly, the generating system and the target CGI program use Perl (Practical Extraction and Report Language) [Wall96] [Sch97] as the programming language. Perl can run on virtually every popular operating system. It also has a powerful DBI (Database Interface) driver module [ANO99] to connect to almost all kinds of database systems. Therefore, our generating system can run under most Unix and Windows environments and the target program is portable to different database systems by installing the appropriate DBI driver. The database system used in our target program is DB2 [Cham96], and the Perl DBI driver used in the example code is for DB2. However, the target program can use the Oracle database system by installing the DBI driver for Oracle, or use the mSQL database system by installing the DBI driver for mSQL, and so on. Little code modification is needed to use a different DBI driver. The remaining uses of the DBI driver are universal. The details of this can be found on page 26. Thirdly, our generating system allows the users to define their tables and queries in a text file according to a specific format. This format can

be understood very easily. It hides most of the programming work so that the user is not required to be a programmer.

Currently, similar software is on the market: Microsoft FrontPage [Tau99]. Basically, FrontPage provides friendly user wizards and templates to enable the users to create their web pages and applications without knowing the back-end programming language. It uses an ODBC (Open Database Connectivity) [Tau99] driver to connect to different database systems. The basic web server for FrontPage is the Microsoft Personal Web Server (MSPWS). If the users want to create sophisticated web applications, they must use another web server other than MSPWS. So the FrontPage server extensions need to be installed to operate with different web servers [Tau99]. FrontPage is very good software for creating web applications. It runs under the Windows operating system, including Windows 95/98, Windows 2000 and Windows NT, but can not run under UNIX operating systems. Our overall system can run under both the Windows and Unix operating systems if Perl and the DBI driver are installed. Thus our system is more flexible than FrontPage in terms of the running environment. Moreover, the FrontPage server extensions need to be installed on the users' Personal Web Server (MSPWS) and the web server the users are using. Sometimes the users must negotiate with the system administrators to install the extra packages. Such an extra installation is not needed in our system.

Overall, FrontPage focuses on providing a general and comprehensive solution for Web design and it is Windows-based. Our overall system is a specific solution to the database queries and it is both Unix and Windows-based.

The next section will describe the use of our system in detail. Some implementation techniques will also discussed.

2. System Description and Implementation

<2.1> System Description from the user view

The purpose of our generating system is to reduce the redundant work for DBAs. The use of the generating system is quite simple. There are three main tasks: define the database and the queries in the text file; run the generating system; and then run the target CGI program on a web server to implement the queries. The most difficult part is the text file definition, but the formats are still easy to follow. The instruction for these three tasks follows.

(1) **Define the database and the queries in the text file.**

From the user's perspective, the text file ought to be as simple as possible so the user can master it quickly. There are five parts in the text file: the database server name; the database name; the name of the user who will create the tables; the table structure description; and the query description. The database server and database names are necessary so our target CGI solution program will know which database server and database to connect to at run time. The table structure description gives the information for creating the tables in the database. Because all of the table creation statements have the same syntax, this description part is given in a natural language format. For the queries, some variable values needs to be bound dynamically. In addition, the number of variables varies. So, data binding has to be used to

integrate the variable values into the queries at run time. Hence, the syntax for the query part should describe the variables, like the first query in Figure 1.

The formats of the text file are designed to represent these ideas as clearly as possible. Figure 1 gives an example of the text file.

```
dbserver richert.cac.psu.edu
user     qxz110
dbname   qxz110
@
table work_order
attri     Customer_name, name,char(10), yes,yes
attri     Bldg/Location, address, char(20),yes,yes
attri     room#, room, char(10), yes, no
attri     phone#, phone, char(15), no, no
attri     budget#, budget, char(5), yes,no
attri     Problem, problem, char(50), no,no
attri     Date_submitted, submit_date, date, yes, no
attri     Date_logged,  log_date, date, yes, no
attri     Date_printed, print_date, date,  no, no
attri     Date_completed, complete_date, date, yes, no
attri     Work_order_number, Work_order_no, integer, yes, no
attri     Technician_name, tech_name, char(20), yes, no
@
table equip_name
attri     Department,  department, char(20), yes, yes
attri     Contact_person, contact_name,  char(20), yes, yes
attri     Equipment_needed, equipment, char(20), yes, yes
attri     Brand_of_equipment, brand, char(20), yes, no
attri     Date_needed, date_needed, date, yes, no
attri     Daytime_phone_no, phone, char(20), no,no
attri     Email_address, email, char(20), no,no
attri     Date_submitted, submit_date, date, yes, no
attri     Equipment_order_number, equip_order_no, char(10), yes, no
attri     Serial_number, serial_no, char(10), yes, no
attri     Model_number, model_no, char(10), yes, no
@
query
# find all of the work orders by the name and room number
& roomno room | Customer_name name
select *
from qxz110.work_order
where name= :name  and room =:room
@
query
#find all the information in the equip_name table
&
select *
from qxz110.equip_name
@                                (continued)
```

```
query
#find all of the work orders by submitted date
& Date_submitted date_submit
select *
from qxz110.work_order
where submit_date= :date_submit
@
query
#find the customer names in  work_order by sumbmitted day
& Date_submitted subdate
select name
from qxz110.work_order
where submit_date = :subdate
@
query
#find all of the work orders
&
select *
from qxz110.work_order
@
query
#find name and submitted date by room number in the work_order table
& room_number rono
select name, submit_date
from qxz110.work_order
where room = :rono
@
query
# find the name of the person who is the contact for the most items of
equipment in a specific department
& department  dept
select contact_name
  from qxz110.equip_name
  where department = :dept
  group by contact_name
  having count(*) >= all (
    select count(*)
    from qxz110.equip_name
    where department = :dept
    group by contact_name)
```

Figure 1. Example text file.

The following paragraph explains the text file's syntax by using Figure 1.

Each part of the definition is separated by the symbol **@**.

For the database server and database names, the format is:

dbserver user_server_name

dbname user_database_name

For instance, the server name is *richert.cac.psu.edu* and the database name is *qxz110* in Figure1. The ID of the user who will create the following tables should also be described here. Since a user can input the values into tables that are created by other users, the account ID will be used to indicate the table's original creator in the table name selection page, as in Figure 4. This is required by DB2. For the same reason, the table name in the query definition part should also be prefixed by the original creator's ID, such as *qxz110.work_order* and *qxz110.equip_name* in Figure 1.

For the table structure, the user needs to give the table name first by the format:

table table_name

and then define each attribute for this table using the syntax:

attri description_name, actual_attribute_name, type, yes/no, yes/no

The first yes/no is to indicate if this attribute can be null or not. If the attribute cannot be null, say yes; otherwise, say no. The second yes/no is to show if this attribute is a primary key or not. For example, there are two tables defined in Figure 1. The first table name is *work_order* and the first attribute is *name* with the description name: *customer_name* . The attribute type is *char(10).* It cannot be null and is a primary key attribute. The *customer_ name* here will be used to give the description for inputting the value for attribute *name* from a web browser later. See Figure 5 for an example. The user should notice that each description name and attribute name must be one word or a single string. If multiple words are used, the user must connect them to form a single string, such as *customer_name.* The table definition ends with the symbol **@**. If the users want to define another table, they simply need to repeat the above definition again. In Figure 1, the second table is *equip_name* with 11 attributes.

For the query description, the whole syntax is:

```
query
# query description sentence
& dynamic variable description line
Actual query statement
```

The format starts with the keyword *query*. Then the users give the description sentence for this query that starts with the symbol # on a new line. This description sentence will be used to index each query on a web page, as in Figure 4. The next line is the variable description. If this query does not need a dynamic variable value, then only the symbol & is given here, as the second and fifth queries in Figure 1. If this query needs one or more dynamic variable values, then the pair of the variable description name and actual name follows the symbol &. The syntax is: & description_name actual_name. Each description name and actual name must be one word or a single string, as with the attribute definition. However, the description name for the variable does not have to match the description name for the corresponding attribute of the same table. The third, fourth and sixth queries in Figure 1 are such queries. If the query has more than one variable, then additional pairs of description name and actual names are used. Each pair is divided by the symbol |. The syntax is

& [description_name actual_name (| description_name actual name)*]

such as:

& description_name1 actual_name1 | description_name2 actual_name2

The first query in Figure 1 is such a query.

The last definition in the query part is the actual query statement. This is given as a standard SQL (Structured Query Language) [Cham96] statement, except that the names of variables can be used. If there is a location where a dynamic variable value is needed, the user should

use the corresponding variable name from the variable description sentence here using the syntax: *:variablename*. There is no space between *:* and *variablename*. The actual query is not completed so far. The value for each variable will be obtained from the web browser and put into the appropriate *:variablename* location in the query during the running of our target CGI program. The query statement will actually be run at that time.

(2) **Run the generating system**.

After the text file is ready, issue the command:

our_system_name text_filename output_script_name create_table_statement_name

For example, suppose our system's name is *final.cgi* (it can be renamed to any name), the text file name is *input*, the output solution name is *output1.cgi*, and the file containing the creating table statement is *output1*, then the command should be:

final.cgi input output1.cgi output1

The *output1* file contains the statements for creating tables if there is any table structure information defined in the *input* file. For instance, the *output1* corresponding to the text file in Figure 1 will be:

```
Connect to  qxz110
CREATE TABLE  work_order ( name CHAR(10) NOT NULL,  address CHAR(20) NOT NULL,
room CHAR(10) NOT NULL,  phone CHAR(15),  budget CHAR(5) NOT NULL,  problem
CHAR(50),  submit_date DATE NOT NULL,  log_date DATE NOT NULL,  print_date
DATE,  complete_date DATE NOT NULL,  work_order_no INTEGER NOT NULL,  tech_name
CHAR(20) NOT NULL,
PRIMARY KEY (name,address) )

CREATE TABLE  equip_name ( department CHAR(20) NOT NULL,  contact_name CHAR(20)
NOT NULL,  equipment CHAR(20) NOT NULL,  brand CHAR(20) NOT NULL,  date_needed
DATE NOT NULL,  phone CHAR(20),  email CHAR(20),  submit_date DATE NOT NULL,
equip_order_no CHAR(10) NOT NULL,  serial_no CHAR(10) NOT NULL,  model_no
CHAR(10) NOT NULL,
PRIMARY KEY (department,contact_name,equipment) )
```

Figure 2 Create table statements.

To send these statements to the DB2 system, issue the command:

```
db2 -f output1
```

Then the tables *work_order* and *equip_name* are created in the database *qxz110*.

This paper describes the database system based on the DB2 database system because our target CGI program is using the Perl DBI driver for DB2. However, it can be extended to other database systems, such as Oracle, by installing the appropriate DBI driver and modifying some environment variables in the *httpd.conf* file of the web server. The reader can consult the DBA or the user's manual of the database system for details of this configuration.

(3) **Run the target CGI program on the Internet**.

A web browser, such as Netscape Navigator or Internet Explorer, is needed to run the target CGI program.

Step 1:

The user should consult with the system administrator to establish which URL should be used to run the generated system. For example, if the system administrator places the CGI program in the web server's directory: cgi-bin/wahls , then we should access the target CGI file *output1.cgi* from the URL: richert.cac.psu.edu/cgi-bin/wahls/output1.cgi where richert.cac.psu.edu is the hostname for the web server.

Step 2:

The first page of the target CGI program will be the user name and password login page, as in Figure 3. The user's user name and password for the account on the database system (such as DB2 in our current system) should be input. See the example in Figure 3.

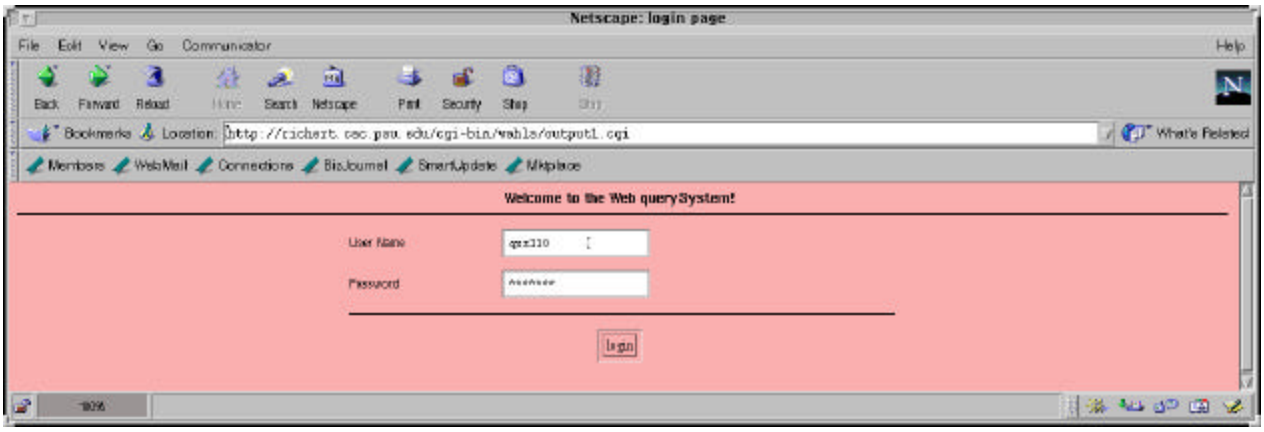

Figure 3. Login page.

Step 3:

If the login succeeds, the selection page for table and query names will appear. The user should then select the table to insert into, or the query to run. Figure 4 is the corresponding selection page for the text file in Figure 1.

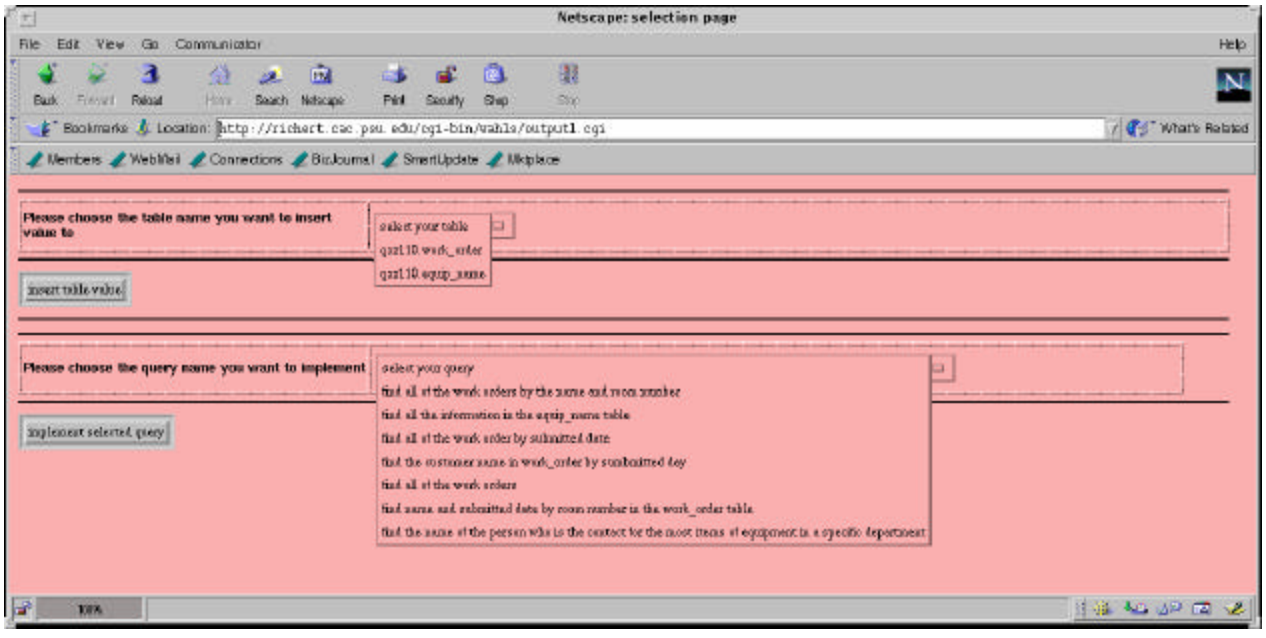

Figure 4. Table and query selection page.

Step 4:

To insert data into a table, the user should select the table name and click “insert table value”. Then the form for collecting the attribute values of that table will appear. The example for the table *work_order* is in Figure 5.

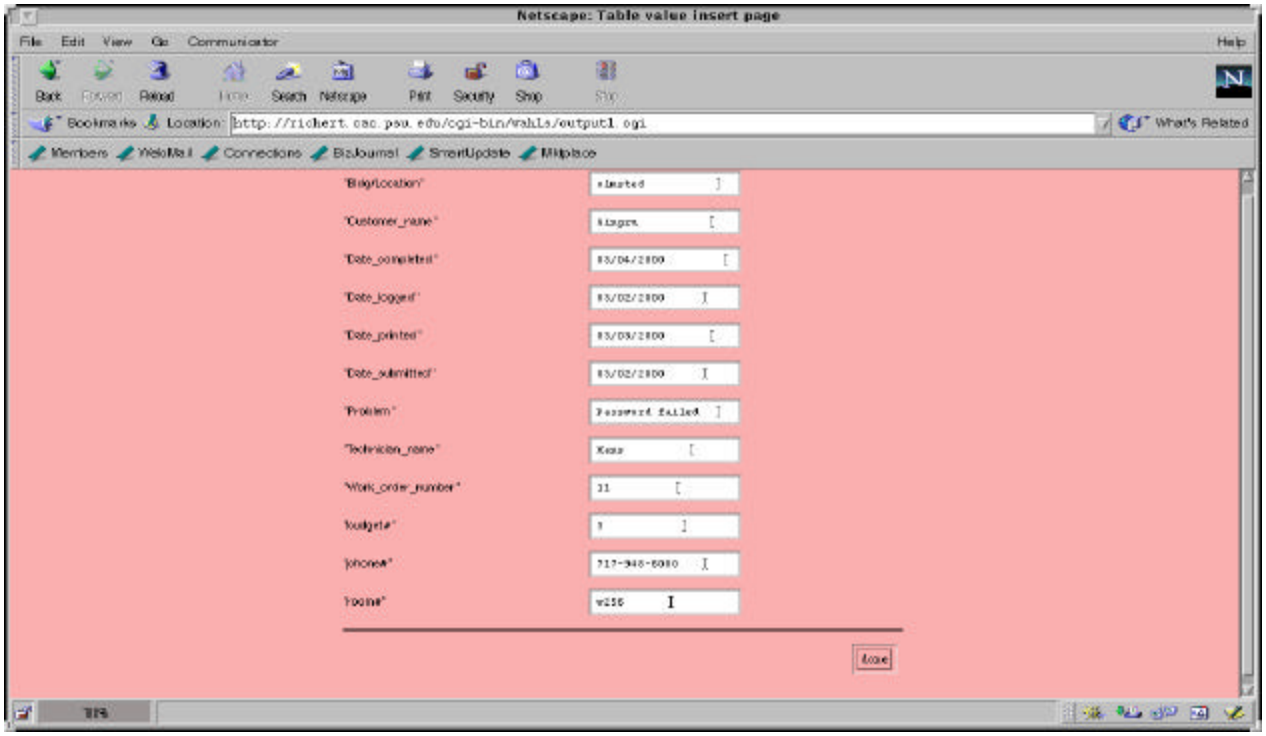

Figure 5. Table value input page.

After finishing inputting the values, the user should click "done.” If the value for each attribute is of the correct type and the new record does not violate a primary key constraint, these values will be successfully inserted into the selected table as a record. Otherwise, a page with the corresponding error message will appear.

Step 5:

To run one of the queries in the query list, the user should select the query name and click “implement selected query” in the selection page, as in Figure 4. If the selected query does

not need any user input, then the query result will display immediately. For example, if we choose the query "find all of the work orders" in the query list in Figure 4, the next page will list its result, as in Figure 6:

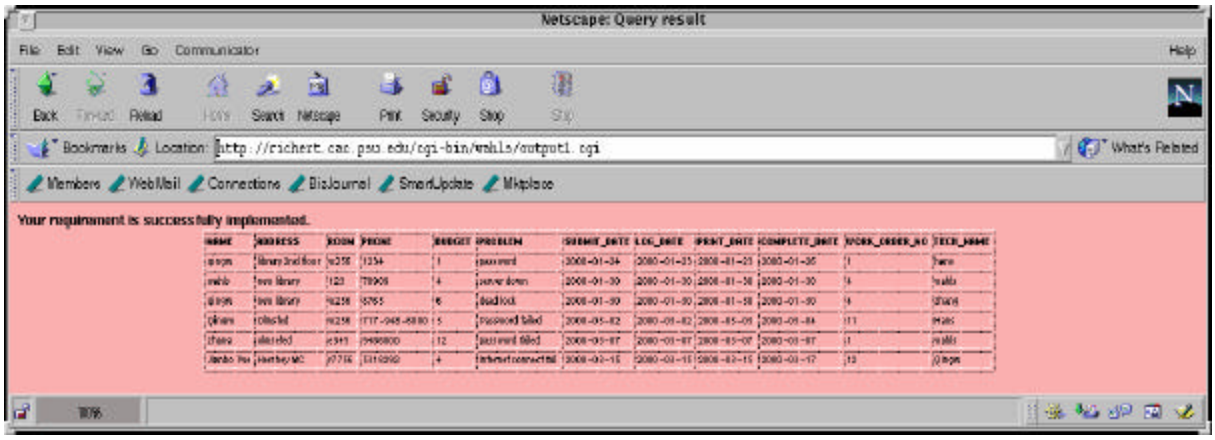

Figure 6. Query result without dynamic variable value input.

If the selected query needs some variable values to be input first, then the variable value entry page will appear. For example, if we choose the query "find all of the work orders by the name and room number" in Figure 4, the variable value entry page will appear as in Figure 7.

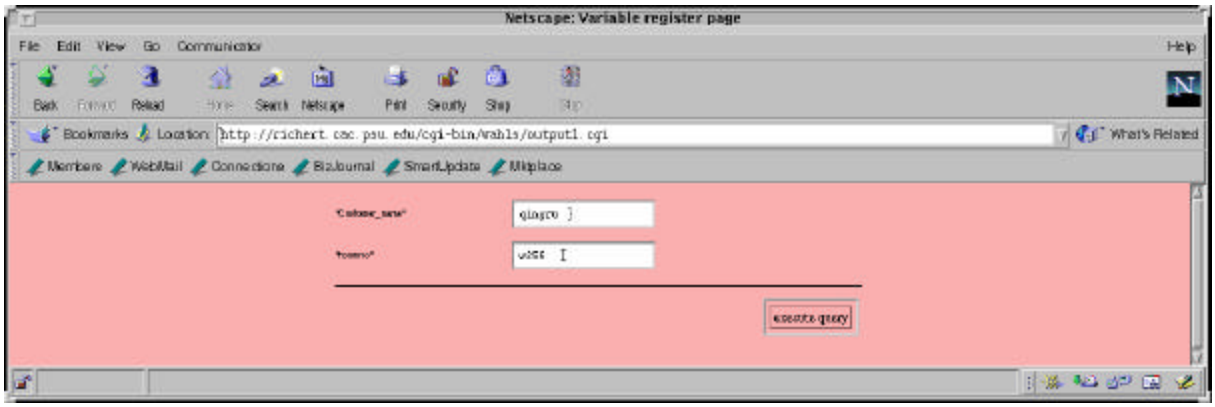

Figure 7. Variable value entry page for query.

If the user inputs *qingru* for name and *W256* for roomno in Figure 7, the query result page will be Figure 8:

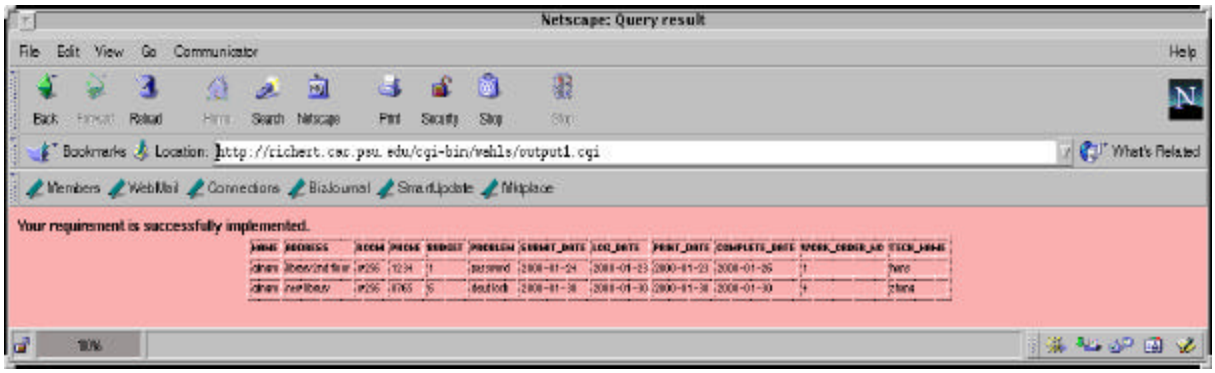

Figure 8. Query result with dynamic variable value needed.

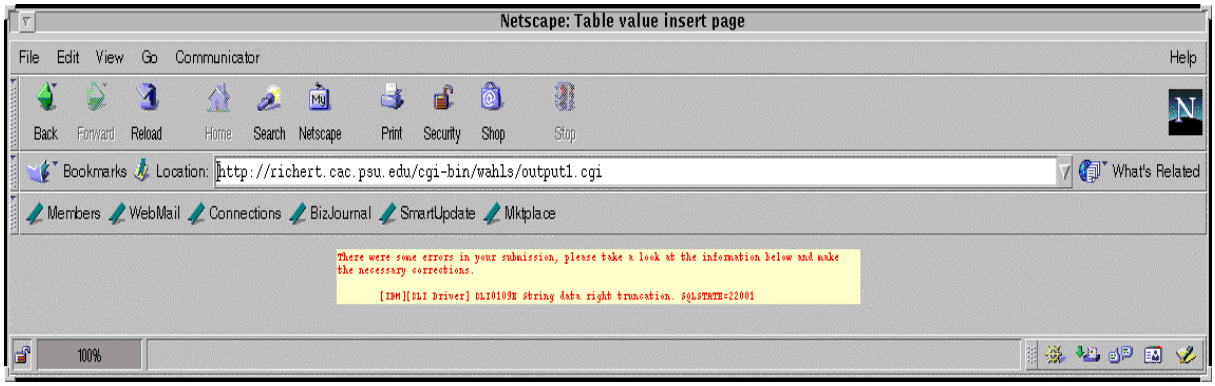

Figure 9. Error message page.

If there is an incorrect value input for some table attribute or dynamic variable, a page for error messages will appear. The user should go back to the previous page, check and fix the errors and click the appropriate operation again. For example, if the user inputs a long string for the "roomno" as in Figure 7, which is larger than the size of attribute *room#*, the error message page will appear as in Figure 9.

This completes the system description from the user perspective. The next part will describe the system implementation from the programmer's view. Implementation ideas and techniques will be described in detail.

<2.2> System implementation from the programmer's view

Our overall system is like a compiler. Reading the text file is similar to the scanning and parsing phases in the compilation process. Then the generating system generates the target CGI program to implement the functions defined in the text file. The target CGI program is equivalent to the target code in the compiler. The processing of our overall system includes three main phases: read the text file, save the information to an organized data structure, and output the web-based target CGI program. Therefore, a language that is able to implement these phases efficiently is very important for our implementation. After careful comparison and consideration, we finally chose Perl [Sher96] as our programming language because it provides all of the powerful functions our implementation needs. Its key features [Cher98] include:

- Powerful text manipulation
  - regular expressions
  - formatted output
- Powerful file management
- Convenient data type management
  - list and hash sorting/searching
  - not strongly typed
  - optional function prototypes
- Rapid development and prototyping
- Compact code for complex operations
- Easy CGI scripting

- Function references

Our implementation mainly uses Perl's powerful text manipulation features to retrieve information from the text file, organize and save information into hash tables, and generate the web-based target CGI program using Perl's CGI module. After our implementation, we feel that Perl is exactly the language we wanted. Since Perl's advantages play an important role in our entire system, we will introduce our implementation idea and techniques by analyzing the features of Perl in the following sections.

Our first implementation part, reading and retrieving information from the text file, is based on the regular expressions of PERL. Regular expressions are a powerful tool for manipulating text. The core of regular expressions is pattern matching. Most modern programming languages offer primitive pattern matching tools through an extra library, whereas Perl's patterns are integrated directly into the language core. As most of the Unix shell interpreters and TCL [Ous94] language do, patterns of Perl provide powerful algorithm accessibility, which is normally only available to computer science scholars.

The common issues of pattern matching are greed, eagerness and backtracking (and how these issues interact with each other) [Chris97].

Greed is the principle that if a quantifier (like *) can match a varying number of times, it will prefer to match as long a substring as it can.

Eagerness is the notion that the leftmost match wins. The engine is very eager to return a match as quickly as possible. For example, the pattern /select/i ..\@/ (find the multiple lines

between the keyword select and @) for finding the query statement in Figure 1 in Perl returns one query statement each time, not all seven query statements**.** The reason is that Perl uses a traditional NFA, a non-deterministic finite state automaton. This kind of matching engine is not guaranteed to return the longest overall match, just the longest, leftmost match. In this sense, Perl's greed is left-to-right directed, not globally greedy. Another kind of matching engine is the DFA, a deterministic finite state automaton. Choosing NFAs or DFAs pattern matching engine depends mainly on the answers to two questions: do the expressions use backreferences, and what needs to be returned (yes/no, range of whole match, ranges of subexpressions). If the language does not need to support backreferences and only needs to return a yes/no answer or the range of the whole match, then it should use DFAs, as in *awk*, *egrep* and *lex*. A DFA is faster and simpler. If the language needs to support backreferences and needs to know which parts of the string were matched by which parts of the pattern, then it should use NFAs, as in *ed*, *regex* and *Perl*. An NFA is slow because it requires potentially exponential run times, but significant performance can be gained when we want to exploit how the particular NFA pattern part is implemented [Chris97] [Frie97].

The last and most powerful feature of Perl's regular expressions is backtracking. For a pattern to match, the entire regular expression must match, not just part of it. Thus, if the beginning of a pattern containing a quantifier succeeds in a way that causes later parts in the pattern to fail, the matching engine backs up and tries to find another match for the beginning part - which is backtracking. This means that the engine will try different possibilities, systematically investigating alternatives until it finds one match that works. However, some pattern matching implementations keep backtracking in case other submatches make the

overall match longer. Perl's matcher does not do that; as long as one possibility works, it stops and uses that, unless something later in the pattern fails [Chris97].

Perl's eagerness and just one possible result of backtracking are very helpful in our implementation. For the same example as above, there are seven queries defined in Figure 1. The pattern to match each query is: /select/i .. /\@/. If the eagerness takes a back seat to greed or the pattern matching implementations keeps backtracking to make the overall match as long as possible, then we always get all of the seven query statements. That will be hard for us to fetch each query's information in detail. Perl's eagerness and backtracking features help with implementing our system easily and efficiently.

Another advantage of Perl that is used in our implementation frequently is its easy string handling. In Perl, strings carry a length, so getting the length is efficient, and adding to the end of the string does not require reading the whole string [Sher96]. The syntax is simple as well. For example, suppose the string holding all of the table information is *eachtable*, and the string holding each attribute value is *attribute*. To append each attribute value to the information for the whole table, the code is : $eachtable .= $attribute. This concatenation implementation is very fast. This is an advantage over the TCL language. In TCL, strings are C strings. Getting the length or adding to the end requires a search through the entire string for a null [Sher96]. Since most of the data structures in Perl (including hash tables that are frequently used in our implementation) are implemented as strings, this leads to some costly handling if using the TCL language.

Another advantage of Perl is its strong support for arrays and hash tables. Perl uses the @ to refer to the array as a whole, and % for the hash table. The size of an array can be easily accessed by @row (if the array name is row). Moreover, a hash table has advantages over a regular array. A regular array uses whole numbers for indices, but the indices of a hash are always strings. Its values may be any arbitrary scalar values, including references. Using references as values, we can create hashes that hold not merely strings or numbers, but also arrays, or objects; or rather, references to arrays, hashes, or objects. The flexibility of the hash table reduces many complex algorithms to simple variable accesses [Chris97].

Hash tables play a core role in our implementation. After reading the text file, we need to keep track of each table with its attribute names. This information will be used in the target CGI program to group and list corresponding attribute names for each selected table. The hash table provides an efficient data structure for these relationships. For each database table's attribute names and description names, we just create a hash table whose name is also the name of table in the database. Within the hash table, the indices are description attribute names, and the value for each index is its actual attribute name. For example, the hash table for saving the attribute information in the database table *work_order* in Figure 1 is:

```
%work_order = qw(
 Customer_name    name
 Bldg/Location    address
 room#    room
 phone#    phone
 budget#    budget
 Problem    problem
 Date_submitted    submit_date
 Date_logged    log_date
 Date_printed    print_date
 Date_completed    complete_date
 Work_order_number    Work_order_no
 Technician_name    tech_name
);
```

The relationship between the table name, the attribute description name and attribute actual name can be described as: $work_order{'room#'} = room (for the attribute room).

Query information is saved by a similar structure, but keeping track of this information is more complicated. For each query, we need to keep track of the description sentence, the query statement, and the dynamic variable names if there are any for this query. Our implementation idea is to create one hash table for all of the query description sentences indexed by the number; then each query statement is saved as a string. The string name for each query statement should contain its corresponding index number in the hash table for the query description sentence. If there are dynamic variables needed for this query, we create a hash table for this query's dynamic variable information as well. This hash table's name should contain the same index number as the corresponding query statement string name. For example, the hash table for all of the query description sentences in Figure 1 should be:

```
%querys = qw(
1 `^find`^all`^of`^the`^work`^orders`^by`^the`^name`^and`^room`^number
2 find`^all`^the`^information`^in`^the`^equip_name`^table
3 find`^all`^of`^the`^work`^order`^by`^submitted`^date
4 find`^the`^customer`^name`^in`^work_order`^by`^sumbmitted`^day
5 find`^all`^of`^the`^work`^orders
6 find`^name`^and`^submitted`^date`^by`^room`^number`^in`^the`^work_order
  `^table
7 find`^the`^name`^of`^the`^person`^who`^is`^the`^contact`^for`^the`^most
  `^items`^of`^equipment`^in`^a`^specific`^department
);
```

The use of "^^" here is to make each description sentence a no-space string, since hash tables use spaces to recognize each pair of key and value. The corresponding query statement for the first query "find all of the work orders by the name and room number" should be:

```
$querystmt1 = " select *
from work_order
where name= :name and room = :room  " ;
```

Also, because there are two dynamic variable values needed to be bound later, a hash table for this query is created as:

```
%queryname1 = qw(
 Customer_name name
 Roomno room
);
```

For a query that does not need any variable values, like the fifth query "find all of the work orders"**,** no **%queryname5** exists in the program. Whenever a query is selected, we can use the code

```
if (!defined(%query))        where

        $query_name =queryname.$query_key;
        %query = %$query_name;
```

to check if such a hash table exists, then implement the appropriate operation. $query_key is obtained from the web users dynamically. The $qeury_name is a hash reference, and the code *%query = %$query_name;* is to dereference this hash reference. From these examples, it is evident that the hash table with its references makes the programming job simple.

After these hash tables are created, we output them into the beginning of the target CGI program. The rest of the code in the target CGI program will be written based on these hash tables by using the references to these hash tables. In this sense, the remaining code

is almost fixed code. The generating program simply outputs the same code for this part each time.

The target program is a web CGI script program. Essentially, CGI is a protocol. The concept of CGI is quite simple: the user provides some information on the web page and the browser sends this information to the web server. The web server passes this information to a particular system. This system processes the information and returns some results to the web server, which then passes the results back to the user's browser. CGI simply defines the means by which the user's data passes from the web server to the processing system and back. The system that processes the user's data can be a database system or a program written in any computer language, as long as these system and languages can run on the server and communicate with the server via CGI. " Perl can 'speak' CGI fluently and happens to be well suited to manipulating data. Leveraged on the flexibility of Perl, almost any conceivable type of processing can be done. Thus, when the Perl program receives the user's data it could follow any number of paths, from retrieving information from a database to constructing and delivering whole new web pages on-the-fly" [Wei99].

In our target CGI program, the processing is done by a DB2 database system since the target system's purpose is to implement the database queries requested by the users from the web browser, and to send the query results back to the web browser. The CGI interaction written in Perl is the bridge between the web server and the DB2 system. Whenever the web server receives the data from the web browser, it sends the data to DB2

and fetches the query result via our target CGI program. What our target CGI program really does is to use Perl's powerful database interface driver, DBI driver(), to connect to and send the query to the database system, and retrieve the query result from the database. In our target program, the DBI driver used is for DB2. That is, we installed the Perl DBI driver on our web server, and issued code in the generated program such as:

```
Use DBI;
Use DBD::DB2::Constants;
Use DBD::DB2;
```

There are also other DBI drivers for Oracle, MSQL, Access, and so on. Our target program could be portable to different database systems easily by installing the appropriate DBI driver and changing the few codes as above [Ano99]. The remaining use of DBI is universal, for example

```
$dbh = DBI->connect($database, $username, $password);  ## connect to database
$sth = $dbh->prepare($stmt);    ## send the query into database
$sth->execute();            ## execute the query
$sth->fetchrow();           ## retrieve the query result
$dbh->disconnect();      ## disconnect from database
```

Thus, our target program is portable due to the portability of the Perl DBI driver.

In terms of the interface of our target CGI program in the web browser, HTML forms are necessary because the web browser needs to collect the user's requests and data and send them to the web server. Normally, a browser can request a document in a number of ways called *methods*, and form values can be encoded in both GET and POST methods. With

the GET method, values are encoded in the URL. That means it can be conveniently bookmarked for canned requests. For example, if we use the GET method for the login page, the URL will be:

http://richert.cac.psu.edu/cgi-bin/wahls/output1.cgi?USER=qxz110&AUTH=password&option=login

where the user name and password are exposed. This situation should be avoided. With the POST method, values are encoded in a different part of the HTTP request that the browser sends to the server. If the form values in the example URL above were sent with a POST request, the user, server, and CGI script all see the URL:

http://richert.cac.psu.edu/cgi-bin/wahls/output1.cgi

Thus, forms that update information on the server, such as mailing in feedback, or querying and updating a database, should use POST [Chris97, 667]. Another reason for this is that client browsers and intervening proxies are free to cache and refresh the results of GET requests, but they may not cache POST requests. GET is only safe for short read-only requests, whereas POST is safe for forms of any size, as well as for updates and feedback responses. Therefore, our target system uses the POST method.

Another thing to mention is that the CGI program is called each time the web server needs a dynamic document target. The CGI program does not run continuously, with the browser calling different parts of the program. Each request for a partial URL corresponding to the CGI program starts a new copy. The CGI program generates a page for that request, then quits. Therefore, the values of variables do not transfer between different pages automatically. If we want to keep a variable state on a different page, we must save it as a HIDDEN field type in the new form's HTML code [Eug96] [Chris97]. This embeds the variable state in the new form. In our target system, to insert values into tables or

implement queries, the different pages such as Figure 5 and Figure 7 need to connect to and disconnect from the DB2 database separately by using the user name and password information from the login page in Figure 3. Therefore, the user name and password information has to be transferred to every page. In order to keep user name and password secure, we save them in a temporary file and passed the file name by the HIDDEN field type in the following form systematically.

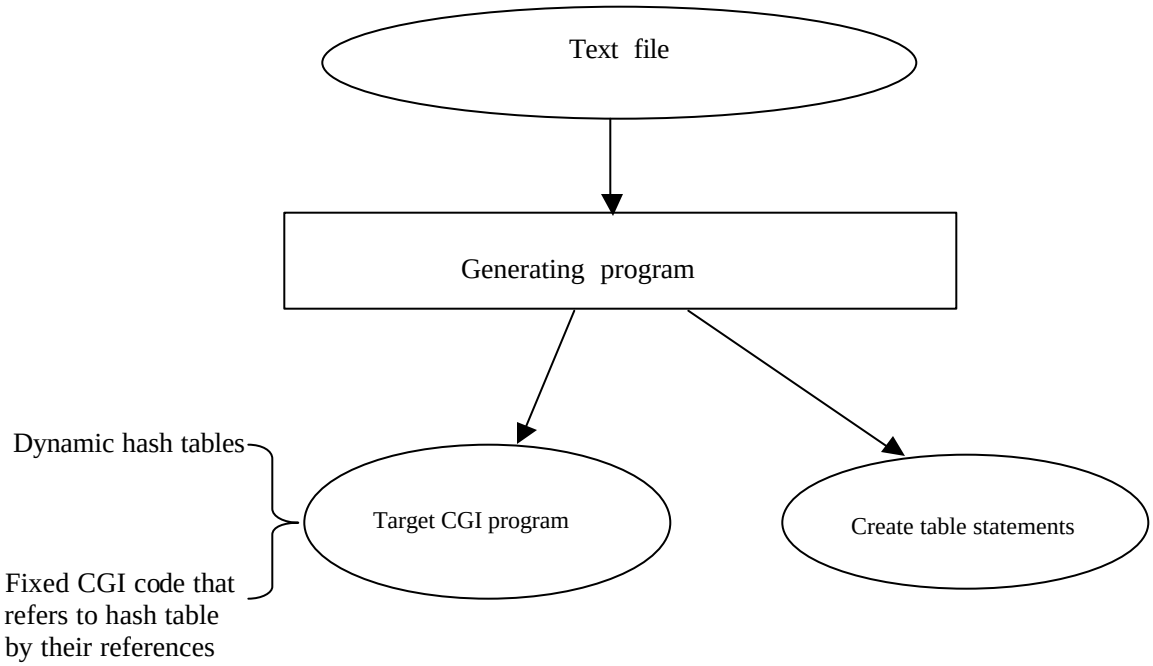

Figure 10. Diagram for our overall system.

After going through each part of our implementation in detail, the overall processing is summarized here:

. Both our generating system and target system are developed in Perl.

. The generating system scans the text file, generates and puts the create table statements into a file, and organizes the dynamic information into hash tables. Then the generating system puts these hash tables into the beginning of the target CGI program. Finally, it writes

the fixed code into the rest of the target CGI program. The hash tables are the core part for generating the dynamic target CGI program. Regular expressions, string manipulation, the DBI driver, and HTML are also tools frequently used in our implementation.
The diagram for our overall system is given in Figure 10.

3. Conclusion

This paper has introduced a specific system that is able to generate a web-enabled solution for database querying, from the user prospective and the programmer view. The input to this generating system is a text file that contains information about tables and queries in a user-friendly format. The output of this generating system is a web CGI program, which provides an Internet interface for implementing the database queries defined in the input text file. The friendly interface of the target web-based program and the easy usage of the generating system allow even non- professional DBAs to administer the database. Hence, our overall system frees DBAs from unnecessary and repetitious work.

However, even though the users can be non-professionals, they still need to call in a database administrator or consultant in certain circumstances. Here are some items that the users can manage themselves, once the database is up and running [Tau99]:

- Adding new tables or new fields to an existing database
- Adding new records to the database
- Running existing queries with different inputs
- Creating queries similar to existing ones

Maintenance items that may require the assistance of a DBA or a consultant include:

- Creating new kinds of queries
- Upgrading the database software
- Setting up the environment variables
- Making regular backups
- Upgrading the hardware of the machine on which the database software runs

As mentioned before, the advantages of our overall system are simplicity, reusability, portability, and web enablement. Among them, web enablement is the most important. The simplicity and reusability are gained from the web enablement due to the intuitive and easy usage of the web browsers, and the reusability of the Internet components.

Currently, web-enablement is an important trend in the business community. A large percentage of Fortune 1000 companies are already being or are converting to Web-Enabled Enterprises (WEE) to improve their flows of electronic transactions via web-based solutions. From the business perspective, effective communications are always the underlying foundation for success in business. Successful organizations must communicate information on many different levels, such as with co-workers, partners, suppliers, and most importantly, with customers. Furthermore, the availability of information is important. The WEE wants to give information about its products and services, and get as much feedback from users as possible [Coen98]. From the IT professional's view, a web-enabled solution integrated with a back-end database system is the right strategy for these business objectives. Web-enabled solutions can provide a platform that enables an organization to work more closely and efficiently with its customers, partners, and suppliers without the barriers of geography and

time differences. The back-end database system can store large amounts of data about products and users' feedback. Therefore, for the IT professional, the main concern here is the available techniques to support this strategy and methods to implement it.

Our overall system introduced in this paper provides an example implementation of this kind of strategy. The database in our system is DB2. The web-compatible programming language used is Perl and its DBI driver for DB2. There are a number of other implementation techniques that can be chosen. For example, the database can be Oracle, mSQL, or Access. The programming language can be Java with a JDBC database driver, Visual Basic with an ODBC driver and so on. The user can also use the Microsoft FrontPage package to generate a simple web-enabled solution without programming language knowledge. Compare to FrontPage, our implementation has its own advantages. It is portable to almost all database systems by installing the appropriate Perl DBI driver. Perl and its DBI driver are widely used in current E-commerce design and web application development.

In short, this paper describes a specific web-enabled solution generating system with its usage and implementation techniques in detail. It gives useful information for web-enablement, an important issue in the current business and IT market, and can benefit web developers and programmers.

References

[ANO99] [anonymous] "DBI for Perl." Course CSCI E-13, Harvard DCE, Fall 99. Online Posting. 12 Dec. 1999 <lab.dce.harvard.edu/extension/cscie13/library/dbi/DBI.html>.

[Cham96] Chamberlin DD. Using the new DB2. San Francisco: Calif. Morgan Kaufmann Publishers; 1996. 682 p.

[Cher98] Chervitz SA. " Perl Tradeoffs." 23 Dec. 1998. Online posting. 6 Mar. 2000 <genome-www.stanford.edu/PerlOOP/bioPerl/oib97/Perl_tradeoffs.html>.

[Chris97] Christiansen T., Nathan T. Perl cookbook. Sebastopol, CA: O'Reilly; 1998. I 757p.

[Coen98] Coen R., Hoogenboom MC. Web-enabled applications programmed on the net: how to become a web-enabled enterprise. New York: McGraw Hill; 1998. 574 p.

[Eug96] Eugnene EK. CGI developer's guide. Indianapolis: Ind. Sams.net; 1996. 497 p.

[Mor98] Moran M. "DBD-DB2-0.68.tar.gz." 1998.Online Posting. 6 June 1999 <http://theory.uwinnipeg.ca/CPAN/data/DB2/DB2.html>.

[Ous94] Ousterhout, J.K. TCL and the TK toolkit. Reading, MA: Addision-Wesley. 1994. 458 p.

[Sch97] Schwartz RL., Christiansen T. Learning Perl. Sebastopol, CA: O'Reilly & Associates; 1997. 269 p.

[Sher96] Sherman A. "Aaron Sherman's Tcl vs Perl Comparison." 1996. Online posting. 6 June 1999 <http://language.Perl.com/versus/asherman-on-tcl.html>.

[Tau99] Tauber DA., Kienan B., Holzschlag ME. Mastering Mocrosoft Frontpage 2000. San Francisco: SYBEX; 1999. 677 p.

[Wall96] Wall L., Christiansen T., Schwartz RL., Potter S. Programming Perl. Sebastopol. CA: O'Reilly & Associates; 1996. 645p.

[Wei99] Weiss A. "Who's Afraid of Perl?" 26 Apr. 1999. Online posting. 20 Mar. 2000 <http://wdvl.com/Authoring/Languages/Perl/PerlfortheWeb/afraid.html>.

```
#############################################################################
#####
#
#
#       A system to generate a simple and reusable web-enabled solution
#
#                         for database queries
#
#                             Qingru Zhang
#
#############################################################################
#####

#!/afs/psu.edu/rs_aix42/usr/local/bin/perl

($input, $output1, $output2) = @ARGV;

if ($input eq '')
  {
   die  "  First paramerter is empty. reading file name is necessary";
  }

if ($output1 eq '')
  {
    die " Second paramenter is empty. writing file name is necessary";
  }

if ($output2 eq '')
  {
    die " Third paramenter is empty. writing file name is necessary";
  }

if (-e $output1)

{
   die " There is already an $output1 file exist. Continue to use this file
name will overwrite the old name. Please give another name.";
}


if (-e $output2)

{
   die " There is already an $output2 file exist. Continue to use this file
name
 will overwrite the old name. Please give another name.";
}
open (INPUT,$input) || die "can't open $input for reading : $! ";
open (OUT,">$output1")  || die "can't create $output1 : $!";
print OUT "#!/afs/psu.edu/rs_aix42/usr/local/bin/perl\n";

$mode = 0755;
chmod $mode, $output1;

while (<INPUT>)  {
   chomp;
   s/ [\cM\cJ]//;
```

```
if  ($_ =~ /dbserver\s+(\S+)/)  # recognize hostname
{
   $hostname = $1;
}
if ($_ =~ /user\s+(\S+)/)  # recognize user ID
{
  $useraccount = $1;
}
if ($_ =~ /dbname\s+(\S+)/)  # recongize database name
{
  $dbname = $1;
  $ENV{"DBI_NAME"} = $1;
}
if (/table/i .. /\@/)   # recognize table definition part
{
 if ($_ =~ /table\s+(\S+)/i)   # recognize table name
   {
    $hashtable .= $1;
    $hashtable .="*";
    $tablename  =$1;
    $eachtable .= $_;
    $eachtable .= " = qw(";
    $eachtable .= "\n";
    $eachtabletype .= $_ . "type";
    $eachtabletype .= " = qw(";
    $eachtabletype .= "\n";
    $tablein .= $_;
    $tablein .= "\n";
    $tablein .="(";
   }
# recognize  attribute name
 if ($_ =~ /attri\s+(\S+),\s*(\S+),\s*(\S+),\s*(\S+),\s*(\S+)/)
   {
    $eachtable .=" ";
    $eachtable .=$1;
    $eachtable .="    ";
    $eachtable .=$2;
    $eachtable .=",";
    $eachtable .="\n";
    $eachtabletype .=" ";
    $eachtabletype .=$2;
    $eachtabletype .="    ";
    $eachtabletype .=$3;
    $eachtabletype .=",";
    $eachtabletype .="\n";
    $tablein .=" ";
    $tablein .=lc($2);
    $tablein .=" ";
    $tablein .=uc($3);
      if ($4 eq "yes")
        {
             $tablein .= " NOT NULL,";
        }
      else {
            $tablein .=",";
           }
```

```
        if ($5 eq "yes")
          {
          $$tablename  .= $2 . " ";
          }
      $tablein .= "\n";
  } #for attri
  } #for table .. @
  if (/query/i .. /\@/)   # recognize query definition part
  {

   if ($_ =~ /\#\s*(\S+)/)  # recognize description sentence
   {
      $querydeso = $_;
      @queryde = split (/\s+/, $querydeso);
      $querydess = join ("`^", @queryde);
      $querydes .= $querydess;
   }
   if ($_ =~ /\&\s*(\S*)/)  # recognize variable definition sentence
    {
     $bindingvar .= $_;
    }
  }
  if (/select/i .. /\@/)  # recognize query statement
    {
      $stmts .= $_;
      $stmts .= "\n";
    }
} # while
      @statment = split (/\@/, $stmts);
      $s=1;
      foreach $statment (@statment)
      {
        chmop;
        s/ [\cM\cJ]//;
        if ($statment =~ /\S+/)
        {
         $querystmt = "\$" . "querystmt";
         $querystmt .=$s;
         $querystmt .= " = \" ";
         $querystmt .=$statment;
         $querystmt .=" \" ";
         $querystmt .=";";
         $querystmt .="\n";
         $s++;
         print OUT $querystmt;  # each query statement
        }
      } # foreach
      @bindingvars = split (/\&/, $bindingvar);
      $index = 0;
      foreach $bindingvars (@bindingvars)
      {
         chmop;
         s/ [\cM\cJ]//;
         if ($bindingvars =~ /\S+/)
         {
           $queryvars = "%queryname";
```

```
            $queryvars .=$index;
            $queryvars .=" = qw( \n";
            @varfields = split (/\|/, $bindingvars);
            $varresult = join("\n", @varfields);
            $queryvars .=$varresult;
            $queryvars .="\n);\n";
           print OUT $queryvars;
           # print dynamic ariable hash table for each query if there exist
one
         }
           $index++;
       }  #foreach
       @hashquerys = split (/\#/, $querydes);
       $querys .= "%querys = qw(";
       $querys .= "\n";
       $m = 1;
       foreach $hashquerys (@hashquerys)
       {
         chmop;
        s/ [\cM\cJ]//;
        if ($hashquerys =~ /\S+/)
        {
         $querys .=$m;
         $querys .=" ";
         $querys .=$hashquerys;
         $querys .="\n";
         $m++;
        }
       } # foreach
       $querys .=");\n";
       print OUT $querys; # print out hash table for query description

      @hashtables = split (/\*/, $hashtable);
      $hashfinal = "%tables = qw(";
      $hashfinal .= "\n";
      $n =1;
      foreach $hashtables (@hashtables)
       {
        chmop;
        s/ [\cM\cJ]//;
        $hashfinal .=$n;
        $hashfinal .="  ";
        $hashfinal .=$hashtables;
        $hashfinal .="\n";
        $n++;
       }
      $hashfinal .=");\n";
      print OUT $hashfinal;
     # print out hash table for table name index

       @eachtablenocomma = split (/,/, $eachtable);
       $eachtableresult  = join(" ", @eachtablenocomma);
       @eachtablehash = split (/table/i, $eachtableresult);
       foreach $eachtablehash (@eachtablehash)
       {
            chmop;
     s/ [\cM\cJ]//;
```

```
    if ($eachtablehash =~ /\S+/)
   {
     $eacht = "%";
     $eacht .= $eachtablehash;
     $eacht .= ");\n";
  print OUT $eacht;
# print out hash tables for each table's attribute description name and
actual name
   }
  } #foreacha


      @eachtabletypenocomma = split (/,/, $eachtabletype);
      $eachtabletyperesult  = join(" ", @eachtabletypenocomma);
      @eachtabletypehash = split (/table/i, $eachtabletyperesult);
      foreach $eachtabletypehash (@eachtabletypehash)
      {
           chmop;
    s/ [\cM\cJ]//;

    if ($eachtabletypehash =~ /\S+/)
   {
     $eachttype = "%";
     $eachttype .= $eachtabletypehash;
     $eachttype .= ");\n";
  print OUT $eachttype;
# print out hash tables for each table's attribute name and type
   }
  } #foreacha

# print out fixed CGI code into output file
print OUT "\$script =\"";
print OUT $output1;
print OUT "\";\n";

print OUT "\$database = \"DBI:DB2:";
print OUT $dbname;
print OUT "\";\n";

print OUT "use DBI;\n";
print OUT "use DBD::DB2::Constants;\n";
print OUT "use DBD::DB2;\n";

print OUT "require 'cgi-lib.pl'; \n ";
print OUT "&ReadParse(*array); \n ";
print OUT " \$errors = 0; \n ";
print OUT " \$errors_list =\"\";\n ";
print OUT "\$ | = 1; # flush output immediately\n ";

print OUT "print \"Content-type: text/html\\n\\n\";\n ";
print OUT "if (\$array{'option'} eq \"\") {\n ";
print OUT " &login; \n ";
print OUT "}\n ";
print OUT "elsif (\$array{'option'} eq \"login\") {\n ";

print OUT "\$username =\$array{'USER'};\n ";
```

```
print OUT "\$auth =\$array{'AUTH'};\n ";
##############################################################################
####################
# for future remote hostname connection, the syntax should be
#
# $attr{                                                                    #
#        hostname=>$hostname
#
#      };
#
# $attr = \%attr;
#
# $dbh = DBI->connect ($database, $username, $auth, $drivername(such as
'DB2' here), \%attr); #
##############################################################################
####################
print OUT "\$dbh = DBI->connect (\$database, \$username, \$auth); \n ";
print OUT "if (!defined(\$dbh)) {\n ";
print OUT "  \$errors++;\n ";
print OUT "  \$errors_list .= \$DBI::errstr; \n ";
print OUT "  &print_top2; \n ";
print OUT "  &print_errors; \n ";
print OUT "} else {\n ";
print OUT "&print_form1; \n ";
print OUT "\$dbh->disconnect(); \n";
print OUT "  }\n ";

print OUT "} elsif (\$array{'option'} eq \"insert table value\") {\n ";
print OUT "   &print_top2; \n ";
print OUT "   &print_form2; \n ";
print OUT "}\n ";
print OUT "elsif (\$array{'option'} eq \"implement selected query\") {\n ";
print OUT "\$query_key = \$array{'QUERY_NAME'}; \n ";
print OUT "\$query_name =queryname.\$query_key; \n ";
print OUT "%query = %\$query_name; \n ";
print OUT " if (!defined(%query)) {\n ";
print OUT "            (\$errors, \$errors_list) = &query_without_variable;
\n ";
print OUT "            if (\$errors > 0) \n ";
print OUT "              {\n ";
print OUT "               &print_top2; \n ";
print OUT "               &print_errors; \n ";
print OUT "              }\n ";
print OUT "           else {\n ";
print OUT "                 &print_thankyou; \n ";
print OUT "                 &print_queryresult; \n ";
print OUT "                }\n ";
print OUT "}  else {\n ";
print OUT "        &print_form3; \n ";
print OUT "        }\n ";
print OUT "}\n ";
print OUT "elsif (\$array{'option'} eq \"done\")   {\n ";
print OUT "      (\$errors, \$errors_list) = &save_data; \n ";
print OUT "       if (\$errors > 0) \n ";
print OUT "          {\n ";
print OUT "           &print_top2; \n ";
print OUT "           &print_errors; \n ";
```

```
print OUT "           }\n ";
print OUT "      else {\n ";
print OUT "           &print_queryresult; \n ";
print OUT "           &print_thankyou; \n ";
print OUT "            }\n ";
print OUT "}\n ";
print OUT "elsif (\$array{'option'} eq \"execute query\") {\n ";
print OUT "         (\$errors, \$errors_list) = &query_with_variable; \n ";
print OUT "            if (\$errors > 0) \n ";
print OUT "              {\n ";
print OUT "               &print_top2; \n ";
print OUT "               &print_errors; \n ";
print OUT "              }\n ";
print OUT "           else {\n ";
print OUT "                &print_thankyou; \n ";
print OUT "                &print_query_withvariable_result; \n ";
print OUT "               }\n ";
print OUT "}\n ";

print OUT "exit 1; \n ";

print OUT
"\###########################################################################\
n ";
print OUT "\############### program subroutines
\###################################\n ";
print OUT "sub login\n ";
print OUT "{\n";
print OUT "print <<EOM  \n";

print OUT "<FORM METHOD=POST ACTION=\$script>         \n ";
print OUT "<html>                    \n ";
print OUT "<head>         \n ";
print OUT "<title>login page</title>  \n ";
print OUT "</head>  \n ";
print OUT "<body bgcolor=\"faafaf\">\n ";

print OUT "<CENTER>\n ";
print OUT "<TR>\n ";
print OUT "<TD COLAPAN=4><B><FONT FACE=\"Arial, Helvetica\">Welcome to the
Web query";
print OUT " System! \n ";
print OUT "</FONT></B></TD>\n ";
print OUT "</TR>\n ";

print OUT "<hr noshade>";
print OUT "<TABLE BORDER=0 WIDTH=\"500\">\n";

print OUT "<TR>\n";
print OUT "<TD><FONT FACE=\"Arial,Helvetica\"><FONT SIZE=-1>User
Name</FONT></TD>\n ";

print OUT "<TD><INPUT TYPE=TEXT NAME=USER VALUE=\$array{'USER'}>\n ";
print OUT "</TD>\n";
print OUT "</TR>\n";
print OUT "<TR>\n ";
```

```
print OUT "<TD><FONT FACE=\"Arial,Helvetica\"><FONT SIZE=-
1>Password</FONT></TD>\n ";

print OUT "<TD><INPUT TYPE=PASSWORD NAME=AUTH VALUE=\$array{'AUTH'}>\n ";
print OUT "</TD>\n ";
print OUT "</TR>\n ";
print OUT "<TR>\n ";
print OUT "<TD COLSPAN=2 align =center><hr noshade><input type=submit
name=\"option\"\n ";
print OUT "value=\"login\">\n ";

print OUT "</TD> \n ";
print OUT "</TR> \n ";

print OUT "</TABLE> \n ";
print OUT "</CENTER>\n ";
print OUT "</BODY>  \n ";
print OUT "</HTML>  \n ";
print OUT "</FORM>\n\n";
print OUT "EOM\n";
print OUT "}\n ";


print OUT "sub print_form1\n ";
print OUT "{\n ";

print OUT "\nprint <<EOM;\n ";

print OUT "<FORM METHOD=post ACTION=\$script>\n ";
print OUT "<INPUT TYPE=HIDDEN NAME=username value=\$array{'USER'}>\n ";
print OUT "<INPUT TYPE=HIDDEN NAME=authname value=\$array{'AUTH'}>\n ";
print OUT "<html>\n ";
print OUT "<head>\n ";
print OUT "<title>selection page</title>\n ";
print OUT "</head>\n ";
print OUT "<body bgcolor=\"#faafaf\">\n ";

print OUT "<hr noshade>\n ";
print OUT "<TABLE BORDER=1>\n ";

print OUT "<TR>\n ";
print OUT "<TD COLSPAN=2 WIDTH=\"30%\"><B><FONT FACE=\"Arial, Helvetica\">
Please ";
print OUT "choose the table name you want to insert value to
</FONT></B></TD>\n ";
print OUT "<TD WIDTH=\"70%\"><SELECT NAME=TABLE_NAME>\n ";
print OUT "<OPTION VALUE=\" \">select your table\n\n";

print OUT "\nEOM\n";

print OUT "foreach \$table (sort keys (%tables)) \n ";
print OUT "{\n";
print OUT "print\"<OPTION VALUE
=\\\"\$table\\\">$useraccount.\$tables{\$table}\\n\";\n";
print OUT " } \n ";
print OUT "\nprint <<EOM;\n ";
```

```
print OUT "</SELECT>\n ";
print OUT "</TD>\n ";
print OUT "</TR>\n ";
print OUT "</TABLE> \n ";

print OUT "<hr noshade>\n ";

print OUT "<TR>\n ";
print OUT "<TD COLSPAN=2 align = left><input type=submit name=\"option\"
value=\"insert table value\">\n ";
print OUT "</TD>\n ";
print OUT "</TR>\n ";
print OUT "<hr noshade>\n ";
print OUT "<hr noshade>\n ";

print OUT "<TABLE BORDER=1>\n ";

print OUT "<TR>\n ";
print OUT "<TD COLSPAN=2 WIDTH=\"30%\"><B><FONT FACE=\"Arial, Helvetica\">
Please";
print OUT " choose the query name you want to implement</FONT></B></TD>\n
";
print OUT "<TD WIDTH=\"70%\"><SELECT NAME=QUERY_NAME>\n ";
print OUT "<OPTION VALUE=\" \" > select your query\n";

print OUT "\nEOM\n";

print OUT "foreach \$query (sort keys (%querys)) \n ";
print OUT "{\n";
print OUT "\$querys{\$query} =\~ s\/\\\`\\\^\/ \/g;\n";
print OUT "print \"<OPTION VALUE =\\\"\$query\\\">\$querys{\$query}\";\n ";
print OUT "}\n";

print OUT "\nprint <<EOM;\n ";

print OUT "</SELECT>\n ";
print OUT "</TD>\n ";
print OUT "</TR>\n ";
print OUT "</TABLE> \n ";

print OUT "<hr noshade>\n ";

print OUT "<TR>\n ";
print OUT "<TD COLSPAN=2 align = center><input type=submit name=\"option\"
value=\"implement selected query\">\n ";
print OUT "</TD>\n ";
print OUT "</TR>\n ";

print OUT "</BODY>\n ";
print OUT "</HTML>\n ";
print OUT "</FORM>\n";


print OUT "\nEOM\n ";

print OUT "}\n ";
```

```
print OUT "sub print_form2\n ";
print OUT "{\n ";
print OUT "\nprint <<EOM;\n ";

print OUT "<FORM METHOD=post ACTION=\$script>\n ";
print OUT "<INPUT TYPE=HIDDEN NAME=edit value=\$array{'TABLE_NAME'}>\n ";
print OUT "<INPUT TYPE=HIDDEN NAME=username2 value=\$array{'username'}>\n
";
print OUT "<INPUT TYPE=HIDDEN NAME=authname2 value=\$array{'authname'}>\n
";
print OUT "<html>\n ";
print OUT "<head>\n ";
print OUT "<title>Table value insert page</title>\n ";
print OUT "</head>\n ";
print OUT "<body bgcolor=\"#faafaf\">\n ";

print OUT "<CENTER>\n ";
print OUT "<TABLE BORDER=0 WIDTH=\"500\">\n ";

print OUT "\nEOM\n ";

print OUT " \$table_key = \$array{'TABLE_NAME'}; \n ";
print OUT " \$table_name = \$tables{\$table_key};\n ";
print OUT "%table = %\$table_name; \n ";

print OUT "foreach \$field (sort keys %table) {\n ";
print OUT "\$v_name = \$table{\$field};\n ";
print OUT "print\"<TR>\";\n ";
print OUT "print\"<TD><FONT FACE=\\\"Arial,Helvetica\\\"><FONT SIZE=-1>";
print OUT" \\\"\$field\\\"</FONT></TD>\";\n ";

print OUT "print\"<TD><INPUT TYPE=TEXT NAME=\\\"\$v_name\\\"
VALUE=\\\"\$array{\$v_name}\\\">\";\n ";
print OUT "print\"</TD>\";\n ";
print OUT "print\"</TR>\";\n ";
print OUT "   }   \n ";
print OUT "\nprint <<EOM;\n ";
print OUT "<TR>\n ";
print OUT "<TD COLSPAN=2 align = right><hr noshade><input type=submit
name=\"option\" value=\"done\">\n ";

print OUT "</TD>\n ";
print OUT "</TR>\n ";

print OUT "</TABLE>\n ";
print OUT "</CENTER>\n ";
print OUT "</BODY>\n ";
print OUT "</HTML>\n ";
print OUT "</FORM>\n ";

print OUT "\nEOM\n ";
print OUT "}\n ";

print OUT "sub print_form3\n ";
print OUT "{\n ";
```

```
print OUT "\nprint <<EOM;\n ";

print OUT "<FORM METHOD=POST ACTION=\$script>   \n ";
print OUT "<INPUT TYPE=HIDDEN NAME=qk value=\$array{'QUERY_NAME'}> \n ";
print OUT "<INPUT TYPE=HIDDEN NAME=username3 value=\$array{'username'}>\n
";
print OUT "<INPUT TYPE=HIDDEN NAME=authname3 value=\$array{'authname'}>\n
";
print OUT "<html>\n ";
print OUT "<head>\n ";
print OUT "<title> Variable register page</title>\n ";
print OUT "</head>\n ";
print OUT "<body bgcolor=\"#faafaf\">\n ";

print OUT "<CENTER>\n ";
print OUT "<TABLE BORDER=0 WIDTH=\"500\">\n ";

print OUT "\nEOM\n ";

print OUT "foreach \$field (sort keys %query) \n ";
print OUT "{\n ";

print OUT "\$variable_name = \$query{\$field};\n ";

print OUT "print\"<TR>\";\n ";
print OUT "print\"<TD><FONT FACE=\\\"Arial,Helvetica\\\"><FONT
SIZE=1>\\\"\$field\\\"";
print OUT "</FONT></TD>\";\n ";

print OUT "print\"<TD><INPUT TYPE=TEXT NAME=\\\"\$variable_name\\\"";
print OUT "VALUE=\\\"\$array{'\$variable_name'}\\\">\";\n";
print OUT "print\"</TD>\";\n ";
print OUT "print\"</TR>\";\n ";
print OUT "} \n ";
print OUT "\nprint <<EOM; \n ";

print OUT "<TR>\n ";
print OUT "<TD COLSPAN=2 align = right><hr noshade><input type=submit
name=\"option\"value=\"execute query\">\n";

print OUT "</TD>\n ";
print OUT "</TR>\n ";

print OUT "</TABLE>\n ";
print OUT "</CENTER>\n ";
print OUT "</BODY>\n ";
print OUT "</HTML>\n ";
print OUT "</FORM>\n ";


print OUT "\nEOM\n ";

print OUT "}\n ";

print OUT "sub query_without_variable\n ";
print OUT "{\n ";
print OUT "  \$stmt =querystmt.\$array{'QUERY_NAME'};\n ";
```

```
print OUT "   \$username =\$array{'username'};\n ";
print OUT " \$auth =\$array{'authname'};\n ";
print OUT " \$dbh = DBI->connect (\$database, \$username, \$auth); \n ";
print OUT " \$sth =\$dbh->prepare(\$\$stmt); \n ";
print OUT " if (\$dbh->errstr) \n ";
print OUT " { \n ";
print OUT "  \$errors++;\n ";
print OUT "  \$errors_list .=\$dbh->errstr; \n ";
print OUT " }  \n ";
print OUT " \$sth->execute();\n ";
print OUT " if (\$sth->errstr) \n ";
print OUT " {   \n ";
print OUT "  \$errors++; \n ";
print OUT "  \$errors_list .=\$sth->errstr; \n ";
print OUT " } \n ";
print OUT "\$sth->finish();\n ";
print OUT "\$dbh->disconnect(); \n";
print OUT "return (\$errors, \$errors_list); \n ";

print OUT "}     \n ";


print OUT "sub query_with_variable\n ";
print OUT "{\n ";
print OUT "  \$stmtstr =querystmt.\$array{'qk'};\n ";
print OUT "    \$stmt =\$\$stmtstr; \n ";
print OUT "    \@each = split (\/ \/, \$stmt);\n ";
print OUT "    \@vararray = (\"\"); \n ";
print OUT "    foreach \$ch (\@each)\n";
print OUT "    { \n";
print OUT "      if (\$ch =\~ \/\\\.\*\\\:(\\S\+)\/g )\n";
print OUT "         {\n";
print OUT "            push (\@vararray, \$1);\n ";
print OUT "         }\n";
print OUT "    }\n";
print OUT " \$stmt =\~ s\/\\\:\\S\+\/\\\?\/g; \n";

print OUT "    \$username =\$array{'username3'};\n ";
print OUT "    \$auth =\$array{'authname3'};\n ";
print OUT "    \$dbh = DBI->connect (\$database, \$username, \$auth); \n ";
print OUT "   \$sth =\$dbh->prepare(\$stmt); \n ";
print OUT "   if (\$dbh->errstr) \n ";
print OUT "   {\n ";
print OUT "   \$errors++;\n ";
print OUT "    \$errors_list .=\$dbh->errstr; \n ";
print OUT "   }\n ";
print OUT "   for (\$id = 1; \$id <= \$\#vararray; \$id\+\+) \n ";
print OUT "   { \n ";
print OUT "       \$var_val = \$vararray[\$id];\n ";
print OUT "       \$values =\$array{\$var_val};\n ";
print OUT "       \$sth->bind_param(\$id, \$values); \n ";
print OUT "   }\n ";
print OUT "   \$sth->execute();\n ";
print OUT "   if (\$sth->errstr) \n ";
print OUT "   {\n ";
print OUT "    \$errors++;\n ";
print OUT "    \$errors_list .=\$sth->errstr; \n ";
```

```
print OUT "   }\n ";
print OUT "  \$sth->finish();\n ";
print OUT "\$dbh->disconnect(); \n";

print OUT "   return (\$errors, \$errors_list); \n ";
print OUT "  }\n ";

print OUT "  sub save_data\n ";
print OUT "  {\n ";
print OUT "     \$username =\$array{'username2'}; \n ";
print OUT "    \$auth =\$array{'authname2'}; \n ";
print OUT "    \$dbh = DBI->connect (\$database, \$username, \$auth); \n ";
print OUT "   if (\$dbi::errstr) \n ";
print OUT "   {        \n ";
print OUT "   \$errors++; \n ";
print OUT "   \$errors_list .=\$dbi::errstr; \n ";
print OUT "   }        \n ";
print OUT "   \$table_key =\$array{'edit'};\n ";
print OUT "   \$table_name = \$tables{\$table_key};\n ";
print OUT "   %table = %\$table_name; \n ";
print OUT "   \$table_name_type = \$table_name . \"type\";\n ";
print OUT "   %tabletype = %\$table_name_type; \n ";

print OUT "    \$sql .=\"INSERT INTO \".\"$useraccount.\".\$table_name; \n
";
print OUT "    \$sql .=\"\(\"; \n ";
print OUT "   foreach \$field (sort keys %table) {\n ";
print OUT "      \$variable = \$table{\$field};\n ";
print OUT "      \$sqlv .=\$variable; \n ";
print OUT "      \$sqlv .=\",\"; \n ";
print OUT "      if ( (\$tabletype{\$variable} =~ /integer/i) ||
(\$tabletype{\$variable} =~ /smallint/i) || ";

print OUT " (\$tabletype{\$variable} =~ /decimal/i) ||
(\$tabletype{\$variable} =~ /double/i) )\n ";
print OUT "   {\n ";
print OUT " \$array{\$variable} =~ s/\'/\'\'/g; ";
print OUT "    \$inputvalue =\$array{\$variable};\n ";
print OUT "   } else {\n ";
print OUT " \$array{\$variable} =~ s/\'/\'\'/g; ";
print OUT " \$inputvalue =\"\'\" . \$array{\$variable} . \"\'\";\n ";
print OUT " }\n ";
print OUT " \$sqlval .= \$inputvalue; \n ";
print OUT " \$sqlval .=\",\"; \n ";
print OUT " }\n ";
print OUT " \$sqlv =~ /(.+)\,/;\n ";
print OUT " \$sqls = \$1; \n ";
print OUT " \$sqlval =~ /(.+)\,/;\n ";
print OUT " \$sqlvals =\$1; \n ";
print OUT " \$sql .=\$sqls; \n ";
print OUT " \$sql .=\"\\\)VALUES\\\( \";\n ";
print OUT " \$sql .=\$sqlvals; \n ";
print OUT " \$sql .=\"\\\)\"; \n ";
print OUT " \$sth =\$dbh->prepare(\$sql); \n ";
print OUT " if (\$dbh->errstr) \n ";
print OUT " {\n ";
print OUT "  \$errors++;\n ";
```

```
print OUT "  \$errors_list .=\$dbh->errstr; \n ";
print OUT "  }     \n ";
print OUT "  \$sth->execute();\n ";
print OUT "  if (\$sth->errstr) \n ";
print OUT "   {\n ";
print OUT "    \$errors++;\n ";
print OUT "    \$errors_list .=\$sth->errstr; \n ";
print OUT "   }\n ";
print OUT "\$dbh->disconnect(); \n";

print OUT "  return (\$errors, \$errors_list); \n ";
print OUT "  }\n ";

print OUT "  sub print_queryresult {\n ";

print OUT " \$dbh = DBI->connect (\$database, \$username, \$auth); \n ";
print OUT " \$sth =\$dbh->prepare(\$\$stmt); \n ";
print OUT " \$sth->execute();\n ";
print OUT " print\"<html>\";            \n ";
print OUT "      print\"<head>\";        \n ";
print OUT "      print\"<title>Query result </title>\"; \n ";
print OUT "      print\"</head>\";             \n ";
print OUT "      print\"<body bgcolor=\\\"#faafaf\\\">\"; \n ";
print OUT "      print\"<center>\"; \n ";
print OUT "      print\"<table border = 1>\"; \n ";
print OUT "      print\"<TR>\"; \n";
print OUT "      for (\$s = 0; \$s <\$sth->{NUM_OF_FIELDS}; \$s++) \n";
print OUT "      { \n";
print OUT "       print\" <TD><B><FONT FACE=\\\"Arial, Helvetica\\\"><FONT
SIZE =1>\$sth->{NAME}->[\$s]</FONT></B></TD>\"; \n";
print OUT "      } \n";
print OUT "      print \"</TR>\"; \n";
print OUT "  while ((\@row = \$sth->fetchrow())){\n ";
print OUT "      print\"<TR>\";\n";
print OUT "   for (\$i =0; \$i <\$sth->{NUM_OF_FIELDS}; \$i++) {\n ";
print OUT "print\"<TD><FONT FACE=\\\"Arial, Helvetica\\\"><FONT SIZE=1>
\$row[\$i]</FONT></TD>\";\n";
print OUT "  } #for\n ";
print OUT "      print\"</TR>\";\n";
print OUT "  } # while\n ";
print OUT "  print\"</table>\";\n ";
print OUT "  print\"</center>\";\n ";
print OUT "  print\"</body>\";\n ";
print OUT "  print\"</html>\";\n ";
print OUT "  \$sth->finish();\n ";
print OUT "\$dbh->disconnect(); \n";

print OUT "  }     \n ";

print OUT "  sub print_query_withvariable_result\n ";
print OUT "  {\n ";
print OUT "    \$dbh = DBI->connect (\$database, \$username, \$auth); \n ";
print OUT "  \$sth =\$dbh->prepare(\$stmt); \n ";
print OUT "   for (\$id = 1; \$id <= \$\#vararray; \$id++) \n ";
print OUT "   { \n ";
print OUT "       \$var_val = \$vararray[\$id]; \n ";
print OUT "       \$values =\$array{\$var_val}; \n ";
```

```
print OUT "       \$sth->bind_param(\$id, \$values); \n ";
print OUT "   }   \n ";
print OUT "  \$sth->execute();\n ";
print OUT "      print\"<html>\"; \n ";
print OUT "      print\"<head>\"; \n ";
print OUT "      print\"<title>Query result </title>\";  \n ";
print OUT "      print\"</head>\"; \n ";
print OUT "      print\"<body bgcolor=\\\"#ffaaff\\\">\"; \n ";
print OUT "      print\"<center>\";  \n ";
print OUT "      print\"<table border = 1>\";\n ";
print OUT "      print\"<TR>\"; \n";
print OUT "      for (\$s = 0; \$s <\$sth->{NUM_OF_FIELDS}; \$s++) \n";
print OUT "      { \n";
print OUT "       print\" <TD><B><FONT FACE=\\\"Arial, Helvetica\\\"><FONT
SIZE =1>\$sth->{NAME}->[\$s]</FONT></B></TD>\"; \n";
print OUT "      } \n";
print OUT "      print \"</TR>\"; \n";
print OUT "  while ((\@row = \$sth->fetchrow())){\n ";
print OUT "      print\"<TR>\";\n";
print OUT "   for (\$i =0; \$i <\$sth->{NUM_OF_FIELDS}; \$i++) {\n ";
print OUT "print\"<TD><FONT FACE=\\\"Arial, Helvetica\\\"><FONT
SIZE=1>\$row[\$i]";
print OUT "</FONT></TD>\"; \n ";
print OUT "  } #for\n";
print OUT "      print\"</TR>\";\n";
print OUT "  } # while \n ";
print OUT "  print\"</table>\"; \n ";
print OUT "  print\"</center>\"; \n ";
print OUT "  print\"</body>\";\n ";
print OUT "  print\"</html>\"; \n ";
print OUT "  \$sth->finish();\n ";
print OUT "\$dbh->disconnect(); \n";

print OUT "  }\n ";

print OUT "  sub print_errors\n ";
print OUT "  {\n ";

print OUT "print <<_ERRORS_;\n ";
print OUT "   <Center><table width=\"500\" border=0
bgcolor=#FFFFCC><tr><td>\n ";
print OUT "   <font face=ARIAL color=Red size=-1>There were some errors in
your submission, ";
print OUT "  please take a look at the information below and make the
necessary corrections. \n ";
print OUT "  <ol>\$errors_list</ol>\n ";
print OUT "  <ol>\$sql</ol>\n ";

print OUT "  </td></tr></table></center>\n";
print OUT "_ERRORS_\n ";

print OUT "  }\n ";

print OUT "  sub print_top2\n ";
print OUT "  {\n ";

print OUT "\nprint <<_END_DOC_;\n ";
```

```
print OUT "  <html>\n ";
print OUT "  <head>\n ";
print OUT "  <title>Table value insert page</title>\n ";
print OUT "  </head>\n ";
print OUT "  </html>\n ";
print OUT "\n_END_DOC_\n ";
print OUT "  }\n ";

print OUT "  sub print_thankyou\n ";
print OUT "  {\n ";
print OUT "\nprint <<EOM; \n ";

print OUT "  <body bgcolor=\"#faafaf\">\n ";

print OUT "  <TR>\n ";
print OUT "  <TD COLAPAN=2><B><FONT FACE=\"Arial, Helvetica\">Your
requirement is";
print OUT "  successfully implemented. </FONT></B></TD>\n ";
print OUT "  </TD>\n ";
print OUT "  </TR>\n ";

print OUT "  </BODY>\n ";

print OUT "\nEOM\n ";
print OUT "  }\n ";

close(OUT) || die " can't close $output1: $!";
open (OUT,">$output2")  || die "can't create $output2 : $!";
print OUT "connect to ";
print OUT $dbname;
print OUT "\n";
   @tables = split (/table/i, $tablein);
   foreach $tables (@tables)
    {
     chmop;
     s/ [\cM\cJ]//;

     if ($tables =~ /\S+/)
    {
      $tables =~ /s*(\S+)\.*/s;
      $currenttable = $1;
      $stmt = "CREATE TABLE ";
      $stmt .= $tables;
      if ($$currenttable ne "" )
       {
        @pkstring =split (/ /,$$currenttable);
        $$currenttable =join(",", @pkstring);
        $stmt .= "PRIMARY KEY (" .$$currenttable . ") ";

       }
      else {
             $stmt =~ /(.*),/s;
             $stmt =$1;
           }
      $stmt .= ")";
      $stmt =~ s/\n/ /g;
   print OUT $stmt; # print out the create table statement to another file
```

```
   print OUT "\n";
    } # if
   } #foreach

close(INPUT) || die "can't close $input: $!";
close(OUT) || die " can't close $output2: $!";
exit 0;
```